Intergral Information Solutions GmbH
Schickardstr 32 • D-71034 • Böblingen • Germany

FusionReactor JDBC Driver Wrapper: User Guide

Doc. Rev. 288, 15 January 2008

Trademarks and Warranties

Feedback

We welcome feedback on all our products and publications. Please e-mail them to support@fusion-reactor.com and we will address them as quickly as possible.

Published in Germany

FR IO / 3.0

Table of Contents

Introduction

Introducing the FusionReactor JDBC Driver Wrapper

The FusionReactor JDBC Driver Wrapper allows developers and administrators to control the interaction between Java and a database. The driver wrapper allows fine-grained metrics and reporting of database activity:

- *Logging of statements which ran against a database*
  This feature is useful to help detect deadlocks, see exactly what `Statements` look like without resorting to manual log output, and to see exactly how your `PreparedStatements` were bound before being run against the database.
- *Row Limiting*
  The integrated row limiter can automatically halt database read activity after a user-specifiable number of rows is reached. This can stop run-away queries before they become a memory and resource problem.
- *Notification and Reminders*
  The driver can optionally notify you when a certain number of rows has been read, and periodically thereafter. Using this feature, you are able to keep a clear overview about the volume of data being processed by Java.

The driver wraps any existing JDBC driver and is able to communicate metric data to FusionReactor for easy perusal in the FusionReactor Administrator. If the driver does not detect FusionReactor running (for instance when running in a standalone Java application), it reports metrics to the standard output stream.

Intended Audience

This technical document is targeted at developers and administrators of standalone and J2EE/ColdFusion applications. It presents the procedure for installing and configuring the FusionReactor JDBC Driver Wrapper to run as a ColdFusion Data Source or as a standalone JDBC data source.

The administrator is expected to have experience with ColdFusion Data Sources, and with how JDBC URLs are constructed.

Limitations

The FusionReactor JDBC Driver Wrapper is intended to help you debug and manage J2EE JDBC queries. It integrates tightly with FusionReactor to provide you with metrics and detailed information about your pages' database activities. Our JDBC Driver Wrapper can't optimize your queries and pages before they're run - but it *can* help you see where the time is being spent, or to locate a 'stuck' query. The FusionReactor JDBC Driver Wrapper also can't totally insulate you from performance or stability issues with underlying drivers.

Some of our customers have experienced stability issues with the standard Macromedia drivers shipped with ColdFusion, for example, and while FusionReactor helped to pinpoint the problem, it was ultimately resolved by trying an alternative driver. For example, many database vendors ship their own JDBC drivers, and they are often very satisfactory for production use.

Installation in ColdFusion

Delivery

The FusionReactor JDBC Driver Wrapper is delivered as an integral part of the FusionReactor product, and is located within the fusionreactor.jar file, which is installed during the FusionReactor setup process. In order to use the driver with ColdFusion applications, you need only to alter the Data Source definition within the ColdFusion administrator.

Wrapping or Adding a New Datasource?

If you are wrapping an existing datasource, you should continue to follow the instructions in this section. If you are starting from scratch, skip ahead to “Creating a New Wrapped Datasource in ColdFusion” on page 9.

Wrapping an Existing Datasource

Underlying all ColdFusion (and indeed all Java) database interaction is a configuration string called a **JDBC URL**. This string tells ColdFusion almost everything it needs to connect to the database: which driver to use, which machine to connect to, which port, and sometimes also the database login and password too.

When using a standard ColdFusion driver, the ColdFusion Administrator application constructs this URL 'behind the scenes', using the details you enter into the form about your server. The FusionReactor JDBC Driver Wrapper requires this URL in order to function. Fortunately, finding it is quite simple.

We recommend the following process, and we've provided a worked example to show you how it works.

1. In the **ColdFusion Administrator**, select the **Server Settings -> Settings Summary** page.
2. This is a long page, but contains the JDBC URL required for the wrapper. The database datasources are outlined in **Data and Services -> Database Data Sources**. If you don't feel like paging down to it, you can also search within that page for the name of your datasource.

   Our test datasource, imaginatively entitled 'testds', looks like this (it's very long and all on one line):

   ```
   jdbc:macromedia:sqlserver://int0006:1433;databaseName=frtest;send
   StringParametersAsUnicode=false;MaxPooledStatements=1000
   ```
3. Note that down (or copy it to your clipboard). Now we're going to add a new wrapped datasource based on this URL.
4. Access the ColdFusion Administrator's **Data & Services -> Data Sources** page. In the **Add New Data Source** panel, enter the name of your wrapped data source. In the **Driver** dropdown box, select **Other**. Click **Add** to go to the details page.
5. In the **JDBC URL** box, enter the first part of our wrapper syntax:

   `jdbc:fusionreactor:wrapper:{`

   .. then paste in the JDBC URL you copied from the Settings Summary page before. Finally, add a final closing brace **`}`**.

   Our wrapped datasource now looks like this (the wrapper syntax has been printed in bold):

   `jdbc:fusionreactor:wrapper:`
   `{``jdbc:macromedia:sqlserver://int0006:1433;databaseName=frtest;sen`**
   `dStringParametersAsUnicode=false;MaxPooledStatements=1000``}`**
6. If you name the datasource explicitly now, FusionReactor will display the name in the “Data Source” column of the JDBC tab in the Request Details page. We'll do that now,

by adding a new parameter to our JDBC URL (in bold):

```
jdbc:fusionreactor:wrapper:
{jdbc:macromedia:sqlserver://int0006:1433;databaseName=frtest;sen
dStringParametersAsUnicode=false;MaxPooledStatements=1000};name=F
RTestDataSource
```

Don't forget the semicolon between the end of the wrapped URL and the **name** parameter.

7. In the **Driver Class** box, enter the name of our wrapper:

   ```
   com.intergral.fusionreactor.jdbc.Wrapper
   ```

8. In the **Driver Name** box, enter something sensible (really it can be anything -- this parameter is never used).
9. In the **User Name** and **Password** boxes, enter your database username and password.
10. Submit this form and test the data source. If it works, rename the old one to something else (“-old”, perhaps) and rename the new wrapped one to the old name.
11. Done!

Creating a New Wrapped Datasource in ColdFusion

This section shows you how to create a wrapped datasource from scratch. It explains how to construct JDBC URLs for ColdFusion's builtin drivers (called the Macromedia drivers, since they were introduced by Macromedia in ColdFusion MX), as well as for existing custom (non-Macromedia) datasources.

Using Macromedia's (DataDirect) ColdFusion Built-in Drivers

If you are using the Macromedia-supplied drivers, you will need to convert your Data Source to an 'other'-type user-specified driver so it can be wrapped by FusionReactor.

1. Create a new Data Source within ColdFusion Administrator as an 'other'-type driver.
2. In the JDBC URL field, enter the original JDBC URL for your data source, wrapped by the FusionReactor URL in curly braces {}. You can additionally name the datasource using the **name** parameter (don't forget to separate the parameter from the wrapped URL with a semicolon). If you name the data source in this way, FusionReactor will display this name in the **Data Source** column of the JDBC tab in the Request Details page. This is useful if you're using multiple data sources.

   E.g. original data source was MS-SQL Server, database 'testdb', named 'MyDataSource'.

   Original JDBC URL:
   `jdbc:macromedia:sqlserver://localhost:1433;databaseName=testdb`

   FusionReactor URL:
   `jdbc:fusionreactor:wrapper:`
   `{``jdbc:macromedia:sqlserver://localhost:`
   `1433;databaseName=testdb`**`};name=MyDataSource`**
3. In the JDBC Driver Class field, enter the name of the FusionReactor JDBC Driver Wrapper class:

   `com.intergral.fusionreactor.jdbc.Wrapper`
4. Enter 'FusionReactor' in the Driver Name field, and appropriate username and password values in their respective fields.
5. Click 'Submit'. ColdFusion will immediately test the driver and provide feedback if necessary. When the FusionReactor driver is loaded, it will report “FusionReactor JDBC: Driver loaded.” to the standard output stream (usually logged to `coldfusion-out.log` or `default-out.log` within `cfmx\runtime\logs`)

Constructing JDBC URLs for other DataDirect Drivers (CFMX/7/8)

While it is not within the scope of this manual to provide exhaustive information on how ColdFusion internally constructs JDBC URLs for its built-in drivers, the following template information (taken from the `neo-query.xml` file within the `cfmx\lib` folder) may help you.

Oracle

Class macromedia.jdbc.MacromediaDriver

```
jdbc:macromedia:oracle://[host]:
[port];SID=[sid];sendStringParametersAsUnicode=[sendStringParamet
ersAsUnicode]
```

MS Access

Class macromedia.jdbc.MacromediaDriver

```
jdbc:sequelink:msaccess://[host]:
[port];serverDatasource=[datasource]
```

PostgesQL

Class org.postgresql.Driver

```
jdbc:postgresql://[host]:[port]/[database]?[args]
```

MySQL

Class org.gjt.mm.mysql.Driver

```
jdbc:mysql://[host]:[port]/[database]?[args]
```

IBM DB2

Class macromedia.jdbc.MacromediaDriver

```
jdbc:macromedia:db2://[host]:
[port];DatabaseName=[database];sendStringParametersAsUnicode=[sen
dStringParametersAsUnicode];[args]
```

MS SQL Server

Class macromedia.jdbc.MacromediaDriver

```
jdbc:macromedia:sqlserver://[host]:
[port];databaseName=[database];SelectMethod=[selectmethod];sendSt
ringParametersAsUnicode=[sendStringParametersAsUnicode]
```

Using a User-Specified Driver ("other") Driver

If you are already using a custom (non DataDirect/Macromedia) driver, simply enclose the existing JDBC URL within the FusionReactor syntax as noted in step 2 above, and change the Driver Class to that of the FusionReactor driver, as specified in step 3.

You will also need to supply the FusionReactor 'driver' parameter in order to allow the FusionReactor JDBC Driver Wrapper to locate the appropriate wrapped driver.

E.g. using the Microsoft JDBC Driver for SQL Server 2005:

Original JDBC URL:
`jdbc:sqlserver://127.0.0.1:1433;DatabaseName=testdb`

Original Driver Class:
`com.microsoft.sqlserver.jdbc.SQLServerDriver`

New wrapped JDBC URL:
`jdbc:fusionreactor:wrapper:{`*`jdbc:sqlserver://127.0.0.1:1433;DatabaseName=testdb`***`};driver=com.microsoft.sqlserver.jdbc.SQLServerDriver`**

New Driver Class:
`com.intergral.fusionreactor.jdbc.Wrapper`

After submitting this form, ColdFusion will immediately test the connection and provide any necessary feedback in the ColdFusion Administrator. If the FusionReactor JDBC Driver Wrapper detects any problems with the underlying wrapped driver or with its own options, it will also provide feedback in the same way.

Any exceptions generated by the FusionReactor JDBC Driver Wrapper are prefixed with an exception ID number, which may be used to look up more information later in this document.

When running as part of J2EE/ColdFusion, and if the FusionReactor JDBC Driver Wrapper detects a running FusionReactor application, any queries run during the course of the request will be reported to FusionReactor for inclusion in the user interface, where appropriate.

Installing Third-Party JDBC JAR Files

In order for the Driver Wrapper to locate the underlying driver, the third-party JDBC JAR driver file (usually delivered as one or more JAR files) must be installed in a location visible to the Driver Wrapper classloader. We recommend either of the following locations beneath your ColdFusion installation folder:

- `CFusionMX7\runtime\servers\coldfusion\SERVER-INF\lib`
- `CFusionMX7\runtime\lib`

Standalone Installation (Servlets etc.)

The driver is packaged within the `fusionreactor.jar` file and you must arrange for this file to be in the JVM classpath when the application runs. The URL syntax is as above. In this configuration, you *must* provide the 'driver' parameter, in order that the FusionReactor JDBC Driver Wrapper can load the appropriate underlying wrapped driver.

Using the FusionReactor JDBC Driver Wrapper

Configuration Options

The FusionReactor JDBC Driver Wrapper is configured exclusively using JDBC URL driver parameters. The following parameters may be specified as name=value pairs, separated by semicolons. No parameters are mandatory. Parameter names are case-insensitive.

driver

Value: Fully-qualified Java class name
Default: no underlying driver will be loaded.
This option instructs FusionReactor JDBC Driver Wrapper to load an underlying wrapped driver.

This is not necessary **only** if the JVM is already aware of the target driver (i.e. it has already been loaded with `Class.forName("...")`). Macromedia's own DataDirect drivers are loaded automatically by ColdFusion, so this option may not be necessary if you are using these drivers. However, if you are using a user-specified driver (having a JDBC URL which does not start with `jdbc:macromedia`), you **must** supply this parameter.

Since the registration of the driver is only ever performed once, regardless of how many connections the driver processes, this parameter can (and should) be specified on all FusionReactor wrapped Data Sources.

If you do not specify this option, and the JVM is *not* aware of the underlying driver, FusionReactor will raise an exception and ColdFusion will not verify the driver.

rowLimit

Value: integer
Default: 0 (disabled).
This option instructs the FusionReactor JDBC Driver Wrapper to limit returned rows to the given value.

After the application has retrieved this number of rows from the result set, FusionReactor will discard any remaining rows.

notifyAfter

Value: integer
Default: 0 (disabled).
This option instructs the FusionReactor JDBC Driver Wrapper to output a notification after 'n' rows have been retrieved for the query.

remindAfter

Value: integer
Default: 0 (disabled).
This option instructs the FusionReactor JDBC Driver Wrapper to periodically output a query reminder every 'n' rows. If **notifyAfter** is specified, FusionReactor JDBC Driver Wrapper will only begin reminding after the notification threshold has been reached.

E.g. notifyAfter=1000, remindAfter=100, actual rowcount 1350.
Notification occurs at row 1000, reminders at 1100, 1200 and 1300.

inhibitReformat

Value: Boolean
Default: false.
When tracking queries, the FusionReactor JDBC Driver Wrapper will reformat them for logging and presentation by attempting to make them fit on a single line.

This allows logs to be viewed more easily, but may hinder developers who are used to seeing queries formatted a certain way (as they are written in a ColdFusion page, for example). Setting this option to 'true' stops FusionReactor JDBC Driver Wrapper reformatting statement text, and allows multi-line presentation in the FusionReactor interface and log.

logToFusionReactor

Value: Boolean
Default: true.
If set to true (the default) and the FusionReactor JDBC Driver Wrapper detects a running FusionReactor instance, it will log the execution of a query to FusionReactors '`jdbc-X.log`' (where 'X' is the current rolling log number).

If this option is enabled and FusionReactor was *not* detected, it has no effect.

interpretObjects

Value: Boolean
Default: true.
If set to true (the default), when a **PreparedStatement** attempts to bind an **Object** type to a positional parameter using one of the **setObject**(...) methods, the wrapper will attempt to interpret the data (for logging and reporting purposes only) by calling the **toString()** method on the object. This value will then be used in the log and FusionReactor administrator, as if the application had called a **setString**(...) method. If the object does not override the default **toString()** method, the default behavior is to return the hash code of the object.

If this parameter is false, the wrapper will use the format "**{OBJECT java.class.name xyz}**" where *xyz* is the **.toString()** representation. This makes it clear that the parameter is of type Object, but is perhaps less easy to read in the log and the Administrator.

name

Value: string
Default: empty
If specified, the driver wrapper will report metrics to FusionReactor with the given name. These names will be reported in the JDBC logfile (or as an empty value if not set). The name will also be reflected in the JDBC tab of the Request Details page, allowing the user to differentiate queries which ran against more than one datasource. This is useful when multiple databases are being used to aggregate results, or when different drivers are being tested.

How To Specify These Options

These options pertain to the FusionReactor JDBC Driver Wrapper, and should therefore be specified **outside** of the curly braces used to wrap the original JDBC URL. Any options which are required by the original JDBC URL should remain **within** the braced section.

Here's an example of a wrapped SQL Server JDBC URL, using the Macromedia driver, to which a couple of FusionReactor JDBC Driver Wrapper options have been added. The material in bold illustrates the additional wrapper syntax:

`jdbc:fusionreactor:wrapper:{``jdbc:macromedia:sqlserver://int0007:1433;`*`databaseName`*`=frtest`**`};notifyAfter=1000;remindAfter=200;inhibitReformat=true;name=DataWarehouse`**

You can see that in this example, the **notifyAfter**, **remindAfter, inhibitReform** and **name** options have all been specified. The **databaseName** option pertains to the Macromedia driver, and are therefore *within* the braced section.

Sample JDBC URLs

Here are a few examples of URLs, wrapped with the FusionReactor Driver Wrapper. As above, the additional material has been printed in bold to make it easy to see.

Oracle

Using the Macromedia driver, with the **notifyAfter** FusionReactor Driver Wrapper option:

```
jdbc:fusionreactor:wrapper:
{jdbc:macromedia:oracle://int0234:1521;SID=testdb};notifyAfter=10
000
```

MySQL

Using the MySQL GJT driver, with the **inhibitReformat** FusionReactor Driver Wrapper option:

```
jdbc:fusionreactor:wrapper:{jdbc:mysql://int0003:3306/webshopdb?
defaultFetchSize=400};inhibitReformat=true;driver=org.gjt.mm.mysq
l.Driver
```

Microsoft SQL Server

Using the Macromedia driver, with the **remindAfter** FusionReactor Driver Wrapper option (we've named this data source too):

```
jdbc:fusionreactor:wrapper:
{jdbc:macromedia:sqlserver://int0007:1433;DatabaseName=frtest};re
mindAfter=500;name=SQLServerDataSource
```

.. and again using the Microsoft SQL Server 2005 – which is not supplied by Macromedia, and so must therefore be explicitly specified as the driver:

```
jdbc:fusionreactor:wrapper:
{jdbc:sqlserver://int0007:1433;DatabaseName=frtest};driver=com.mi
crosoft.jdbc.sqlserver.SQLServerDriver
```

... and as a quick reminder, the FusionReactor Driver Wrapper driver class is:

```
com.intergral.fusionreactor.jdbc.Wrapper
```

Interpreting Log Data

When outputting data to the JDBC log file (which can be found in FusionReactor's instance log directory), the FusionReactor JDBC Driver Wrapper outputs a number of fields which can be used to debug JDBC transactions and derive statistics about how the system is using database resources.

The JDBC log file is space-delimited, with text fields (which may contain spaces) enclosed with double-quotes. We have had no trouble importing this data into Microsoft Excel and OpenOffice Calc, as well as Microsoft SQL Server using Data Transformation Packages.

Log Fields

The following list describes the meaning of each field. The list describes the fields in left-to-right order. For field sources listed as 'FusionReactor', this field may be empty **if** the request in which the query ran has no associated FusionReactor tracked request.

Calendar Date

Value: YYYY-MM-DD
Source: Wrapper

Specifies the calendar date on which the log message was raised.

Time

Value: HH:MM:SS
Source: Wrapper

Specifies the 24-hour time at which the log message was raised.

Epoch time

Value: long millisecond
Source: Wrapper

Specifies the exact epoch time (millisecond offset from midnight on January 1st 1970 UTC) at which the log message was raised.

Fusion Request ID

Value: long integer
Source: FusionReactor

Specifies the FusionReactor request ID within whose execution this JDBC interaction occurred.

Thread

Value: String
Source: Wrapper

Specifies the name of the thread in which this JDBC interaction occurred.

Client IP

Value: dotted quad IP address
Source: FusionReactor

Specifies the IP of the client for whom this request is running.

HTTP Method

Value: HTTP 1.X Method (GET / POST / HEAD etc.)
Source: FusionReactor

Specifies the HTTP method of the request which caused this JDBC interaction

URL

Value: Full or Partial URL
Source: FusionReactor

Specifies the URL which caused this request to run. If FusionReactor is tracking complete URLs (which machine name) this will be a full URL. If not, this will be the path element.

Log Message Type

Value: One of METRIC, NOTIFICATION or REMINDER
Source: Wrapper

Specifies the type of this message. *METRIC* reports the completion of a JDBC interaction, NOTIFICATION specifies that a Notification threshold has been reached on the size of the result set, and REMINDER specifies that a result set size reminder interval has been reached.

Execution Start Time

Value: long millisecond
Source: Wrapper

The start time in milliseconds from the epoch datum (see **Epoch Time** above) at which the JDBC interaction began (i.e. the time at which the statement was transferred to the database driver for execution)

Execution End Time

Value: long millisecond
Source: Wrapper

The time at which the underlying database driver finished executing the statement.

Result Set Close Time

Value: long millisecond
Source: Wrapper

The time at which the result set was closed by the J2EE application (e.g. ColdFusion etc.). This interval between this time and the Execution Start Time is useful as the total processing time for the query, including database execution time and the time taken for the J2EE application to fully read and process the result set.

Execution Elapsed Time

Value: long millisecond
Source: Wrapper

The time taken to execute the statement on the database (computed from Execution Start and End times)

Result Set Elapsed Time

Value: long millisecond
Source: Wrapper

The time taken between sending the statement to the underlying driver for execution, and the J2EE application actually closing the result set (computed from the Execution Start time and the Result Set Close Time)

Rows Read

Value: long
Source: Wrapper

Specifies the maximum number of rows read by the J2EE application. If the statement is not a DQL command (select etc.), but rather is DML/DDL/RIGHTS (insert/update, drop/alter/create, revoke/grant etc.) this value will be 0.

Is Prepared Statement

Value: boolean
Source: Wrapper

Specifies whether this statement was prepared in advance of its execution.

Is Row Limited

Value: boolean
Source: Wrapper

Specifies whether the Row Limiter activated to stop the query.

Datasource Name

Value: String
Source: Wrapper

Specifies the name of the datasource (specified by the JDBC 'name' parameter). Blank if the name was not specified.

Statement

Value: SQL String
Source: Wrapper

Specifies the statement which was run during this interaction. Any whitespace formatting in the original statement is flattened to allow the statement to appear on one line. If the interaction was a batch execution, individual statements are delimited by [[and]] strings.

Stack Elements

Value: Comma-separated list of Strings
Source: Wrapper

If FusionReactor is configured to record stack traces, this field contains a comma-separated list of stack frames, recorded when the query completed. If the debug information is available, this field can be used to locate the exact line in a script or Java program which caused the interaction.

URL Parameters

Value: String
Source: FusionReactor

Contains the parameters which were present on the URL associated with the request in which this statement is running.

Message

Value: String
Source: Wrapper

For NOTIFICATION or REMINDER log messages, this field contains the text of the notification or reminder.

Prepared Statement: Positional Bind Parameters Replacement Strings

During presentation of the statement text for Prepared Statements, the FusionReactor JDBC Driver Wrapper attempts to replace the bind parameter placeholder (“?”) with an appropriate representation of the bound value for that parameter. This is not always possible, for example with binary streams. The following table details the replacement.

Textual replacements in italics have no feasible representation and are inserted as literals. All other types are converted to their string representation for output.

Parameter Type	PreparedStatement Method	Text Representation
SQL Array	setArray(...)	*{SQL ARRAY}*
ASCII Stream	setAsciiStream(...)	*{ASCII STREAM}*
Binary Stream	setBinaryStream(...)	*{BINARY STREAM}*
Big Decimal	setBigDecimal(...)	String representation
Blob	setBlob(...)	*{BLOB}*
Boolean	setBoolean(...)	String representation
Byte	setByte(...)	String representation
Bytes	setBytes(...)	String representation: comma-separated list of up to 256 bytes
CharacterStream	setCharacterStream(...)	*{CHARACTER STREAM}*
Clob	setClob(...)	*{CLOB}*
Date	setDate(...)	String represenation using Date.toString() to format
Double	setDouble(...)	String representation
Floating Point Number	setFloat(...)	String representation
Integral Number	setInt(...)	String representation
Long Integral Number	setLong(...)	String representation
Null	setNull(...)	*{NULL}*
Object	setObject(...)	String representation **if** option **interpretObjects** is true (or not specified) **otherwise:** “{OBJECT java.class.name xyz}” where *xyz* is the .toString() representation
Reference	setRef(...)	*{REF}*
Short Integral Number	setShort(...)	String representation
String	setString(...)	String representation
Time	setTime(...)	String representation
Timestamp	setTimestamp(...)	String representation

Parameter Type	PreparedStatement Method	Text Representation
Unicode Stream	setUnicodeStream(...)	*{UNICODE STREAM}*
Uniform Resource Locator	setURL(...)	String representation

A Note On SQL Server Select Methods

In this edition of the JDBC Driver Wrapper manual, we have removed references to the Macromedia/Microsoft SQL Server driver option **selectMethod**. This option controls how the driver and the database supply rows to your application, and can have an impact on memory usage and JDBC processing speed. We want you to make an informed decision about whether to apply this option, and this section will provide an explanation about how this option works.

When You Can Use This Option

This option is applied as a driver parameter in the JDBC URL for **Microsoft SQL Server datasources**. It can be used with the Microsoft JDBC Driver (2000 and 2005), and the built-in Macromedia driver. Since it is a parameter to the underlying JDBC driver, it must be supplied within the wrapped part of the URL:

Using the Macromedia Driver:

```
jdbc:fusionreactor:wrapper:
{jdbc:macromedia:sqlserver://int0007:1433;DatabaseName=frtest;SelectMethod=cursor};remindAfter=500;name=SQLServerDataSource
```

Using the Microsoft SQL Server 2005 JDBC Driver:

```
jdbc:fusionreactor:wrapper:
{jdbc:sqlserver://int0007:1433;DatabaseName=frtest;SelectMethod=cursor};driver=com.microsoft.jdbc.sqlserver.SQLServerDriver
```

Direct and Cursor Selection Modes

If this option is not specified in the JDBC URL, it defaults to **direct**.

Direct Selection Method

When the driver is operating in direct mode, select statements are processed like this:

- Application performs an SQL select.
- JDBC driver transfers the request to the database.
- Database performs the select.
- **Database transfers the complete result set back to the driver**.
- Driver makes each row available to the application.

Cursor Selection Method

Here is the same process when the driver operates in cursor selection mode:

- Application performs an SQL select.
- JDBC driver transfers the request to the database.
- Database performs the select.
- **Database opens a cursor, and sends the first batch of rows back**.
- Driver makes the first batch of rows available.
- Application eventually requests a row which wasn't in the first batch.
- Driver asks the database for more rows.
- **Database uses the cursor to see where it was in the results, and transfers the next batch.**
- .. etc., until the application closes the query or the database exhausts the cursor.

Pros and Cons

Direct mode transfers all the data at once. The data is then stored in the driver's memory space.

Pros

- Very fast for the application to iterate through rows because all data is immediately available.
- Useful for fast batch work where speed is preferable over low memory usage.

Cons

- Large result sets will consume a lot of memory. You must account for this memory when configuring your J2EE (ColdFusion) application server. You must provide enough margin for the size of the largest result set to be requested.
    - Repeatedly opening large result sets may cause Java to perform major garbage collections often, causing the application to become less responsive.
    - Concurrently opening large result sets may cause memory to be exhausted, leading to continuous garbage collection as Java attempts to reclaim memory, and possible crashes.
    - Be aware that the total memory requirement for a given result set is **twice the size of the result set**. When the database runs the query, it must first buffer the results in its own memory space, before transferring all the data to the client.
- Long network delay when query is run while all data is transferred. For databases running on the same server, this delay is shorter but not insignificant, since the TCP/IP transport is still used.
- Large time and memory overhead if not all rows are processed by the application.
    - If the application only processes the first few rows, the transfer and storage of the entire result set is very inefficient.
- Concurrent requests for result sets from the database will cause separate connections to be opened to the database.

Cursor mode transfers data in batches transparently to the application. The complete resulting row set for each query is stored in the database. The J2EE (ColdFusion) application which uses the JDBC driver only stores one batch (typically a small fraction) of rows. The driver takes care of requesting new batches from the database as they are needed.

Pros

- Small memory requirement in the client application, since only one batch of rows is stored.
    - Memory demand burden is placed on the database, which can typically manage it more efficiently than J2EE applications. If the database is running on a dedicated machine, the memory requirement is then transferred to that machine, leaving the J2EE (ColdFusion) sever with more memory to allocate to the application.
- Result set processing can begin much more quickly since there is a much smaller initial delay before the driver makes rows available.
- Repeatedly opening large queries, running large concurrent queries, or only processing the first few rows of a large result set is comparatively cheap, since only the first batch will be transferred and stored in the client.

Concurrently-running queries may be interleaved over the same connection.

Cons

- Total time taken to retrieve an entire result set is slightly higher than in direct mode, since the driver must retrieve several batches of rows.

Caveats for non-ColdFusion JDBC Environments

This section is for engineers using SQL Server JDBC drivers in a J2SE/J2EE environment which is not ColdFusion; e.g. where the driver, connection etc. is obtained manually using Java code.

This caveat applies only to MS SQL Server 2000 JDBC Driver and DataDirect/Macromedia SQL Server Drivers.

This problem does not occur with the MS SQL Server 2005 JDBC Driver.

Connection Commit Status Restrictions in Direct Mode

You may see the following JDBC exception:

> **"Can't start a cloned cloned connection while in manual transaction mode".**

When operating in direct mode, only one active SQL statement (including 'select' statements) can be open over a `java.sql.Connection`.

If a second or subsequent statement is opened while any result sets or statements are still active on the first connection, the connection is transparently cloned by the driver, and the statement is run over the new connection.

If the auto-commit status of the connection has been changed from the default `auto-commit=true` to `auto-commit=false` (that is, all transactions will be explicitly committed or rolled back), the cloned connection may have uncommitted (dirty) data present. In this case it is not safe to start a new statement, and the driver issues the exception above.

To alleviate this situation, one or more the following measures may be taken:

1. Switch to cursor mode by appending `selectMethod=cursor` to your JDBC URL.
2. Switch to auto-commit mode.
3. Ensure that only one statement is active on a connection by closing all statements and result sets prior to opening new ones.
4. Ensure that all connection operations are synchronized against multiple access in a multi-threaded environment.

Furthermore, several operations may cause the driver to internally create a second statement and attempt to run it on the same connection (e.g. updating certain types of result set), producing the same exception.

References

Microsoft KB article 313181 refers: http://support.microsoft.com/kb/313181

Exception Catalog

In almost all cases, the FusionReactor JDBC Driver Wrapper will propagate all SQL Exceptions upwards, allowing them to be transparently handled by an application. In certain cases, FusionReactor JDBC Driver Wrapper can originate its own exceptions.

These exceptions have an ID number placed in square brackets within the message string. This ID number can be used to locate more information in the following table.

ID	Example	Explanation
1	Driver options must be name=value pairs	Indicates an error with the wrapper driver parameters. Check all options have values.
2	value for driver option '(*option*)' was neither 'true' nor 'false' (*value*)	The valid values for this option are either 'true' or 'false'. The actual read value is provided in the message. Check the option specified has a valid value.
4	driver option '(*option*)' is unknown	The supplied option was not known. The rogue option is given in the message. Check the option for typographical errors.
5	couldn't parse value for driver option (*option) (value)* as a number.	The given driver option requires a number as its value, and FusionReactor couldn't parse the given value as a number. The option and the rogue value are given in the message. Check the value to make sure it can be parsed as a number and has no alphabetic characters.
6	driver class (*classname*) could not be found and loaded.	The Java class specified with the 'driver' option could not be found. Check the class or jar is within the application classpath, and the class name is fully qualified (if required)
7	Could not find a wrapped JDBC URL within the passed string. The wrapped URL must be within braces {}.	The Driver Wrapper could not find a wrapped URL within the JDBC URL. Check the syntax of the whole URL and make sure the original URL is specified within braces {...}.
8	Soft kill active for this thread	This thread has been marked for Soft Kill by FusionReactor. The Driver Wrapper will not proceed with any further database activity.
9	URL supplied did not conform to JDBC URL specification. Please check the syntax.	The Driver Wrapper could not reliably decipher the supplied JDBC URL, because it probably contained a syntax or typographical error. Please check it carefully by hand.